ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO EN INFORMÁTICA

Título del proyecto:

IMPLEMENTACIÓN DE UN SISTEMA AUTOMÁTICO DE LOCALIZACIÓN DE REGIONES FACIALES EN VÍDEO POR WEBCAM

Ioritz Cia Ulacia

Miguel Pagola Barrio

Pamplona, 15 de Febrero de 2010

Agradecimientos:
Gracias a los compañeros del despacho y a los profesores por la ayuda y el apoyo prestados.
Gracias a mi familia y amigos por el apoyo y la paciencia que han tenido.

Índice general

Índice de figuras

Capítulo 1

INTRODUCCIÓN

1.1. Seguimiento

El seguimiento es la monitorización del comportamiento, actividades o de información cambiante de generalmente personas y a menudo, de una forma oculta. En la mayoría de los casos se trata de observar a individuos o grupos por parte de alguna organización. En el ámbito de este proyecto generalmente requiere de cámaras de vigilancia y de equipo informático que grabe y/o analice las grabaciones.

Las cámaras de vigilancia tienen el cometido de vigilar un área. Se conectan a un dispositivo de grabación, a una dirección IP o la grabación se envía a un monitor al que está atendiendo un vigilante de seguridad. Hasta hace poco estos equipos eran caros y requerían personal para atenderlos, pero últimamente el precio de estos equipos ha descendido y no es necesario contratar personal de vigilancia dependiendo de la tarea para lo que se quiera. Generalmente el tipo de equipo que se utiliza para este fin es un circuito cerrado de televisión (CCTV closed-circuit television en inglés). Esta es una tecnología que usa cámaras de vídeo para transmitir la señal que estas graban, a un determinado receptor o receptores. Difiere de otras tecnologías como puede ser la televisión convencional, en que no solo el personal autorizado puede acceder a esas grabaciones.

Aunque anteriormente se hable de vigilancia de personas, se puede utilizar el seguimiento con otros propósitos. Uno de estos propósitos puede ser la vigilancia de los procesos industriales, puede resultar extremadamente útil en zonas en las que se necesita vigilancia, pero que por riesgos de salud o de seguridad una persona no pueda estar físicamente en ese lugar. Como pueden ser casos en la industria química o nuclear. También puede utilizarse en este ámbito para automatizar procesos de medición de calidad. Otro cam-

po en el que puede resultar útil es en la monitorización del tráfico. Se puede monitorizar el tráfico para evitar aglomeraciones de vehículos en una zona, para vigilar que los conductores respeten las normas de circulación o para dar avisos de accidentes. También se usa para vigilancia en transportes como pueden ser los metros en las grandes ciudades, donde el operario del metro no es capaz de ver toda la zona alrededor de los vagones. Aunque como con casi todas las tecnologías puede usarse con fines criminales como por ejemplo, grabar los códigos secretos de la gente cuando acceden a un cajero automático.

Aunque las ventajas que tiene son muy buenas, este tipo de tecnologías no está exenta de detractores. Las razones para por las que no agrada a mucha gente son la pérdida de la privacidad, el uso del seguimiento como control social o el acceso ilegal a las grabaciones de los sistemas de vigilancia entre otras.

Una vez que se tiene el equipo de cámaras, el siguiente paso a dar es analizar los vídeos que el equipo graba. El análisis de las secuencias se realiza de forma automática mediante software. Para que no sea necesario adquirir y mantener un equipo de almacenamiento con gran capacidad y por lo tanto caro, solo se almacenan las secuencias en las que se produce algún movimiento, ya que no es necesario vigilar un área en la que no ocurre nada. Se puede realizar el análisis a la vez que se graba o se puede realizar sobre un vídeo ya grabado y almacenado. La primera opción es más costosa ya que normalmente se requiere que el análisis se realice en tiempo real y esto requiere un software optimizado y un equipo con mayores prestaciones. Estos dos requerimentos hacen que el sistema sea más caro.

Hoy en día el seguimiento o la vigilancia no se realizan únicamente desde las cámaras de vídeo, también existen otros disposotivos. Existen máquinas que analizan características biométricas y que no son cámaras conectadas a ordenadores, como por ejemplo escáneres de huellas dactilares o de retina, máquinas que analizan el ADN entre otras. También hay características biométricas que se miden con cámaras y software como patrones faciales, patrones en la forma de caminar, etc. Otra técnica para realizar seguimiento es la minería de datos, en la que no es necesario tener cámaras para observar los comportamientos de una persona. Consiste en analizar datos de personas para poder extraer información sobre ellas. Otro método de seguimiento puede ser el de rastrear dispositivos que pueda llevar una persona para saber su localización, por ejemplo pequeños dispositivos de radiofrecuencia, localizadores G.P.S o teléfonos móviles.

En los últimos 10 años el uso de cámaras de vigilancia y de sistemas informáticos que las controlan y tratan sus secuencias, se ha visto incrementado considerablemente. Los gobiernos mundiales invierten grandes cantidades de

dinero para establecer sistemas como los que se ha mencionado anteriormente en sus ciudades, carreteras y edificios gubernamentales con el fin de ampliar la seguridad en estos.

1.2. Identificación del color de la cara

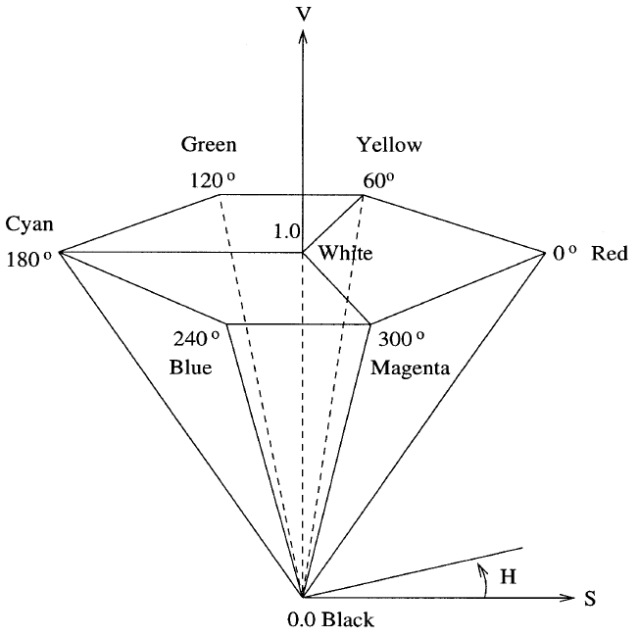
La investigación de técnicas de detección y reconocimiento de caras humanas, es un campo activo del área de la visión por ordenador. Como bien se puede observar en la anterior afirmación hay dos pasos distinguibles en este proceso. El primero es la detección de la cara en la imagen y el segundo es el reconocimiento automático de la cara mediante extracción de características de la misma. Gran parte de la investigación en este campo se ha realizado sobre el paso del reconocimiento facial, sin embargo, el primer paso es crucial para que el segundo pueda realizarse correctamente y para que al final todo el proceso de identificación de una persona en una imagen pueda llevarse a cabo con éxito. Además, no es el único proceso en el que la localización de caras es de gran utilidad. Existen numerosos procedimientos en los que la detección de la cara humana en la imagen es de gran utilidad. Por poner un ejemplo, la eliminación de ojos rojos en software de tratamiento de imágenes.

Existen varios modos de localizar caras humanas en imágenes. Varios de estos métodos se basan en la detección de la cara utilizando información sobre su forma o movimiento. En este proyecto se va utilizar el color como característica principal para detectar la cara. No obstante, esto no es suficiente y una vez seleccionadas las partes de la imagen según el color, se utilizará la forma como característica secundaria para poder eliminar aquellas partes que sean de un color parecido al de la cara pero que no pertenezcan a la cara humana.

El objetivo es detectar el rango de color que representa mejor la cara humana. Como se va a trabajar en color, se debe identificar el rango de color en cada uno de los canales. Existen varios sistemas de color, (RGB, HSV, CIE, L*u*v*) cada uno tiene sus ventajas y sus desventajas y se debe escojer el que mayores ventajas y menores desventajas ofrece. Según el estudio que realizan los autores en el artículo en el que se basa este proyecto, HSV es el sistema de color que mejores características ofrece a la hora de realizar detección de caras por color.

Generalmente el color es representado mediante el sistema de color RGB. Este sistema es muy adecuado desde el punto de vista del hardware tanto para adquirir como para mostrar las imágenes. Sin embargo, no es el sistema más adecuado para la descripción de la percepción del color. El sistema HSV, es el que más se asemeja a la forma en la que la visión humana percibe el

color. Así como el sistema RGB se basa en la mezcla de colores lumínicos y es algo que todo el mundo conoce, la representación del color en el sistema HSV no. Este sistema se basa en mezclar el tono (H de hue), la saturación (S de saturation) y el valor (V de value). Con estos tres valores se forma un poliedro, que se puede observar en la siguiente figura en el que se hallan todos los colores posibles. La combinación de valores funciona de la siguiente manera. El componente H determina el tono del color y su rango oscila entre 0^o y 360^o, la componente S la saturación (la cantidad de tonalidad grisacea del color) y su rango oscila entre 0 y 1 y la componente V el valor (la cantidad de brillo del color) y su rango oscila entre 0 y 1. Siendo así, hay ciertos casos particulares, por ejemplo si la componente V es igual a 0 da igual que valor tengan las otras dos, el color resultante es negro. O si la componente S vale 0 y la componente V vale 1 da igual que valor tenga la componente H, el valor resultante será blanco.



En el artículo se basan en un estudio de color para obtener un poliedro dentro de la figura del sistema HSV, que delimita los rangos de valores que en cada una de las componentes H, S y V se quieren admitir como pertenecientes a color de piel. A este poliedro se le llama Poliedro Principal (PP Principal Polyhedron). Estos son los rangos aceptados de cada componente según el estudio.

$T_{H1} = 340^o \leq H \leq T_{H2} = 360^o \; y \; T_{H3} = 0^o \leq H \leq T_{H4} = 50^o$

$S \geq T_{S1} = 20\,\%$

$V \geq T_{V1} = 35\,\%$

Aunque es suficiente con definir este poliedro, en el artículo definen un segundo poliedro Poliedro Secundario (SP Second Polyhedron) para dar más precisión al resultado del algoritmo, los rangos aceptados de cada componente del segundo poliedro son los siguientes.

$T_{H1} = 340^{o} \leq H \leq T_{H2} = 360^{o}$ *y* $T_{H3} = 0^{o} \leq H \leq T_{H4} = 50^{o}$

$T_{S2} \leq S \leq 20\,\%$

$V \geq T_{V1} = 35\,\%$

1.3. Propósito

El objetivo de este proyecto, es realizar la implementación de una plataforma en la que se ubique la cara de una persona en un video reproduciéndose en tiempo real. Primero, se debe realizar la instalación de una tarjeta capturadora de video en el equipo destinado para el proyecto. Se utilizarán cámaras para grabar el video deseado y mediante la tarjeta instalada se obtendrá el video deseado. Después, se debe realizar la implementación de la aplicación. Esta tarea consiste en dibujar un rectángulo delimitando los límites de las caras de las personas o en dejar en blanco toda la imagen salvo la parte que pertenece a las caras de las personas. Así, una vez se obtiene el resultado, se puede utilizar la parte delimitada para realizar tratamientos de imagen sobre ella.

Capítulo 2

ALGORITMO

2.1. Pseudocódigo

El siguiente es el proceso que se ha seguido para tratar la imagen. El proceso consta de dos fases, la primera fase es un módulo de procesado del color en dos fases y la segunda fase es un módulo de análisis de forma y color que se implementa en una tercera fase que es la final.

La primera fase consta de tres bloques tal y como se puede observar en la figura 2.1. En el primer bloque de esos tres, se debe realizar la segmentación utilizando el poliedro principal. En este bloque se extraen los píxeles pertenecientes al poliedro principal y se le pasan al siguiente bloque. El siguiente bloque es el del filtrado escalar, esta parte no se ha implementado, el cálculo realizado por esta parte se sustituye por un valor estático que ha demostrado dar unos resultados buenos y además al no tener que realizar los cálculos se gana en eficiencia. En el tercer bloque se le aplican a la imagen un filtro de la mediana y un proceso de eliminación o relleno de píxeles. Este bloque se realiza para la obtención de objetos de un determinado tamaño en adelante. El resultado de esta fase se le pasa al segundo bloque de la segunda fase.

La segunda fase también consta de tres bloques. En el primer bloque se debe realizar la segmentación utilizando el segundo poliedro. En este bloque se extraen los píxeles pertenecientes al segundo poliedro y se le pasan al siguiente bloque. El siguiente bloque es el de combinar los píxeles del primer poliedro con los del segundo. Es a este bloque al que se le pasa el resultado de la primera fase, con este y con el resultado del bloque anterior se realiza la combinación. En el tercer bloque se realizan un filtro de la mediana, un proceso de eliminación o relleno de píxeles y operaciones morfológicas. El filtro de la mediana y el proceso de eliminación o relleno se utilizan con el mismo

propósito que en la fase anterior. Las operaciones morfológicas se realizan con el propósito de rellenar o eliminar píxeles. Pueden faltar los píxeles de las cuencas de los ojos. Como ya se ha comentado anteriormente en la implementación no se ha utilizado un segundo poliedro. Al tener que cambiar los umbrales del primer poliedro para que en el laboratorio funcionen bien, se ha conseguido realizar una buena segmentación con un único poliedro, por lo que no se ha implementado las operaciones de esta fase que implican al segundo poliedro.

Por último está la tercera fase, en la que se realiza un análisis de forma y color. En esta fase se tiene como entrada la imagen resultante de las operaciones morfológicas y se realiza un análisis de las características de los objetos como son la forma, la simetría y el posicionamiento del objeto dentro de la imagen. En función de estas características se determina si el objeto es una cara o no.

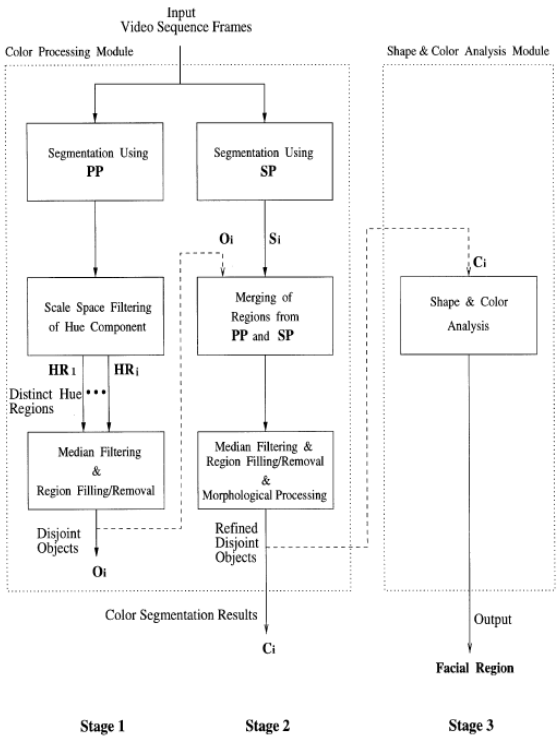


Figura 2.1: Esquema de todo el algoritmo.

A continuación se explica este proceso dividido en los pasos más importantes y se detalla cada uno de los pasos con mayor detalle que en este apartado.

2.1.1. Imagen

Como se ha dicho anteriormente, se desea trabajar con el sistema de color HSV, pero como también se ha dicho el sistema de color del hardware suele ser RGB. En este caso, el sistema de color de la cámara web que se utiliza no es ninguna excepción y utiliza RGB. Por lo tanto, el primer paso a realizar es la conversión de la imagen obtenida por la cámara de sistema RGB a sistema HSV.

Para realizar la conversión de RGB a HSV existen unas fórmulas que según los valores en el sistema RGB, asigna unos valores en el sistema HSV. La primera fórmula indica como obtener el valor del canal H de un píxel con los valores de los canales RGB de ese mismo píxel.

$V \leftarrow max(R, G, B)$

Lo mismo para el canal S.

$S \leftarrow \frac{V - min(R,G,B)}{V}$ $\quad si\ V \neq 0$ o $S \leftarrow 0$ en otro caso.

Lo mismo para el canal H.

$H \leftarrow 60(G - B)/S \quad si\ V = R$ o
$H \leftarrow 120 + 60(B - R)/S \quad si\ V = G$ o
$H \leftarrow 240 + 60(R - G)/S \quad si\ V = B$

Si H $<$0 entonces H=H+360.
Se tiene que $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

No obstante, en OpenCV existe una función para hacerlo automáticamente. A la función `cvCvtColor` se le introduce una imagen en un sistema de color y devuelve otra imagen en otro sistema de color.

En OpenCV se pueden guardar las imágenes con 8, 16 y 32 bits. Dependiendo del número de bits con los que se guarde la imagen los valores de HSV variarán.

Guardando la imagen con 8 bits surgen algunos problemas, ya que no se pueden alcanzar valores mayores que 255 ($2^8 = 256$) y H va de 0 a 360. Estos son los cambios que sufren los valores guardando con 8 bits.

$V \leftarrow 255V$
$S \leftarrow 255S$
$H \leftarrow H/2$

Guardando la imagen con 16 bits los valores sufren las siguientes conversiones. Actualmente este modo no se encuentra disponible.

$V \leftarrow -65535V$

$S \leftarrow -65535S$
$H \leftarrow -H$

Guardando la imagen con 32 bits los valores no sufren conversiones.

$V \leftarrow V$
$S \leftarrow S$
$H \leftarrow H$

2.1.2. Segmentación PP

En la sección anterior se ha hablado sobre los umbrales que se obtienen del estudio de color realizado en el artículo. Estos son los umbrales que se deberían utilizar, pero se han probado y en el emplazamiento de la cámara no funcionan bien. Debido a problemas de iluminación en el sitio de pruebas, se han establecido otros umbrales que proporcionan un mejor rendimiento. De esto se hablará más extensamente en el apartado de problemas y soluciones.

En este apartado del algoritmo, se emplean los umbrales definidos para realizar una selección de píxeles en la imagen de entrada, de modo que se obtienen los pixeles de tonalidad carnosa, descartando todos los demás. Así el primer paso estaría dado, ya que falta descartar las partes de la imagen que tengan tonalidad carnosa pero que no sean una cara. Al identificar los píxeles deseados se cambian a color negro y los demás a color blanco. La figura 2.2 es un ejemplo.

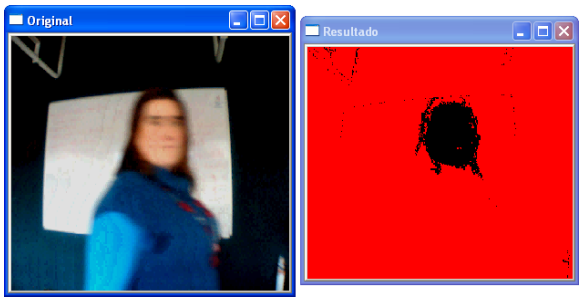


Figura 2.2: Ejemplo de la segmentación del color carne.

2.1.3. Filtro de la Mediana y Eliminacion/Relleno

El filtro de la mediana es una de las técnicas no lineales más utilizadas para el procesado de imágenes, tiene diversos usos y es muy utilizado en la eliminación de ruido en imágenes siendo particularmente efectivo para este propósito. Es un filtro no lineal, tanto en los filtros lineales como en los no

lineales se recorre la matriz pixel a pixel. En el caso de los filtros lineales se aplica la convolución entre una submatriz de píxeles con una matriz de coeficientes, que determina el comportamiento del filtro. Sin embargo, en los filtros no lineales, la operación no es la convolución, en este caso se aplica un algoritmo sobre la matriz de píxeles.

En el caso del filtro de la mediana, este algoritmo consiste en sustituir el pixel del centro de la submatriz por el del valor de la mediana de todos los píxeles de la submatriz. Para ello se escogen ventanas N x N con N impar para tener claro cuál es el pixel central. Es importante elegir bien el tamaño de la ventana, ya que un valor para N pequeño implica que no se elimine bien el ruido y un valor alto implica la distorsión de la imagen.

En este algoritmo se utiliza con el propósito de suavizar las siluetas de los objetos obtenidos en el apartado anterior y para eliminar el ruido que pueda haber en la imagen. No obstante, aunque elimina o rellena muy bien píxeles sueltos no es efectivo a la hora de eliminar regiones pequeñas de píxeles o rellenar pequeños huecos. La función de OpenCV para el uso de filtros es `cvSmooth`. En la figura 2.3 se puede observar como se disminuye el ruido.

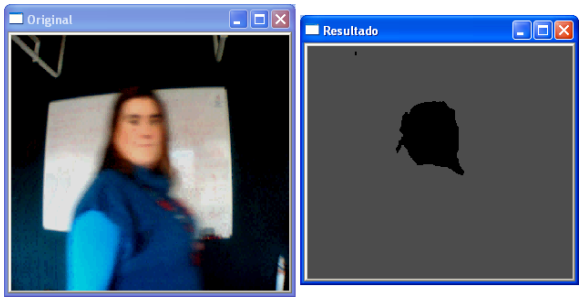


Figura 2.3: Ejemplo del filtro de la mediana.

Como el filtro de la mediana no es efectivo a la hora de eliminar regiones pequeñas de píxeles o rellenar pequeños huecos se realizan las operaciones de eliminación y relleno. Las operaciones de eliminación y relleno lo que hacen es contar el número de píxeles del contorno de un objeto de la imagen. En función de si es mayor o menor que un umbral fijado elimina el objeto o rellena el hueco encontrado.

2.1.4. Operaciones Morfológicas

Las operaciones morfológicas se utilizan en este proyecto para rellenar o eliminar píxeles. Pueden faltar los píxeles de las cuencas de los ojos. Los dos operadores morfológicos que se utilizan en este proyecto son el cierre y la apertura, que resultan de la composición de las operaciones de dilatación y

erosión.

La dilatación consiste en aumentar el número de píxeles del objeto. La dilatación de un objeto X con un elemento estructural B se define como un conjunto de puntos x tales que B_x (la translación de B cuando el origen es x) forma una intersección no vacía con x.

$$X \oplus B = \{x : B_x \cap X \neq \oslash\}$$

La erosión consiste en disminuir el número de píxeles del objeto. La erosión de un objeto X con un elemento estructural B se define como un conjunto de puntos x tales que B_x (la translación de B cuando el origen es x) se incluye en X.

$$X \ominus B = \{x : B_x \subset X\}$$

El cierre es la composición de la erosión sobre la dilatación y la apertura la composición de la dilatación sobre la erosión. La operación de cierre se utiliza para rellenar los pequeños huecos.

$$X^B = (X \oplus B) \ominus B.$$

La operación de apertura para eliminar los pequeños grupos de píxeles o los canales finos.

$$X_B = (X \ominus B) \oplus B.$$

Estas dos operaciones mantienen el tamaño y la forma de los objetos.

De esta forma solo quedarán objetos de un tamaño mayor que los objetos que puedan ser ruido. Uno de los objetos que quede después de estas operaciones, será el que pertenece a la zona facial que se desea localizar.

En este proyecto no se ha utilizado ninguna función de OpenCV para realizar este apartado, las operaciones se han programado. En la figura 2.4 se puede apreciar como queda la imagen después de la aplicación de estos operadores.

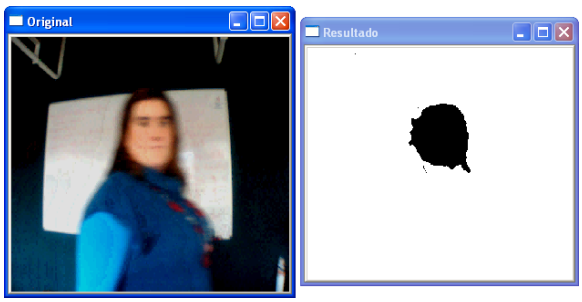


Figura 2.4: Ejemplo de la aplicación de las operaciones morfológicas.

2.1.5. Análisis de forma y color

Una vez se tiene una imagen con los objetos que han quedado, lo que se debe hacer es identificar cuál de esos objetos es la cara. Para este fin se tienen en cuenta características de la cara como son la forma, la simetría y el posicionamiento de la región facial dentro de la imagen. En este proceso se proponen las siguientes características o primitivos a tener en cuenta. En el artículo se trabaja para la localización de caras en videoconferencias por lo que las características están influenciadas por este hecho.

Desviación respecto el valor medio de la tonalidad del tipo de piel de las diferentes categorías. El valor medio de la tonalidad para los tipos de piel diferentes varía entre seres humanos y depende de la raza, del género y de la edad de la persona. Particularmente, en la tonalidad los valores de la piel están dentro de un rango específico para todas las categorías de tipos de piel. La tonalidad media de los tipos de piel diferentes forma un rango que representa la tonalidad más probable para los tonos de piel del ser humano. La desviación del valor de la tonalidad esperada respecto el rango definido indica su semejanza con un color de tono de piel.

Ratio de aspecto de la cara. La silueta de la cara humana tiene una forma más o menos definida y aunque sea diferente en cada persona, la proporción que guardan la altura y anchura de la cara humana en todas las personas es parecida, por lo que se puede determinar un rango. Si el objeto que se está tratando mantiene la proporción entre altura y anchura dentro de ese rango puede que sea una cara, sino no.

Orientación vertical. La localización de una cara en una imagen depende mucho de la posición de la cámara y del ángulo que ésta describa con las caras de las personas que van a ser grabadas. En secuencias de vídeo de videoconferencia se presupone lo siguiente. La cabeza no se muestra inclinada de forma que no hay oclusión de cara, la cabeza no se muestra con un ángulo de rotación muy grande de forma que la simetría de la cara respecto del eje vertical se deforma ligeramente.

Posición relativa de la región facial en la imagen. Por la misma razón que en el apartado anterior es más probable que la cara esté situada en el centro de la imagen.

Para cada objeto se miden cada una de estas características. Cada una de estas características se miden mediante funciones de pertenencia. Para unificar los criterios de las características se utiliza un operador de agregación compensativo. Este es el que se ha escogido.

$$\mu_c = (min_{j=1}^{m}\mu_j)^{(1-\gamma)}(max_{j=1}^{m}\mu_j)^{\gamma}$$

Se ha escogido $\gamma = 0{,}5$ para dar la misma importancia al mínimo que al máximo, por lo que queda así.

$$\mu_c = ((min_{j=1}^{m}\mu_j)(max_{j=1}^{m}\mu_j))^{0{,}5}$$

En la figura 2.5 se puede apreciar como queda la imagen después de finalizar el análisis de la forma y el color.

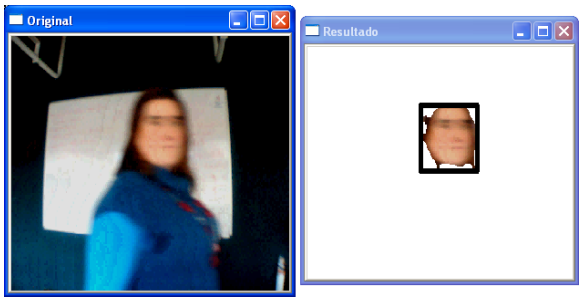


Figura 2.5: Ejemplo del análisis de la forma y color.

Capítulo 3

IMPLEMENTACIÓN

3.1. Integración de Visual Studio C++ y OpenCV

En este tutorial se explica cómo hacer que Visual Studio C++ trabaje con la librería OpenCV. Primero se deben tener ambos instalados. Una vez instalados, se deben seguir los siguientes pasos.

3.1.1. Vincular librerías

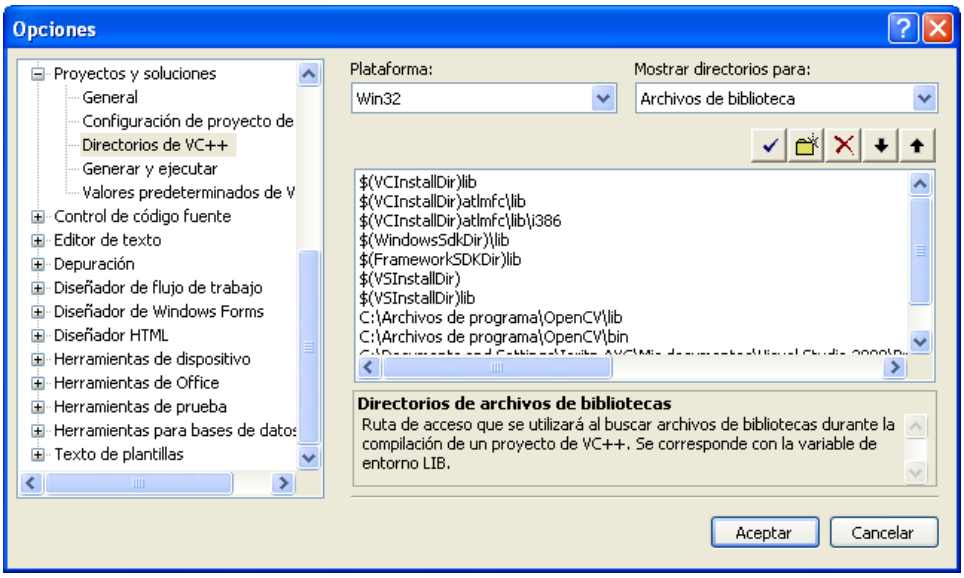
Para incluir librerías de OpenCV sin necesitar ubicarlas en la carpeta de cada proyecto se debe añadir al PATH la dirección de las librerías de OpenCV dentro de la carpeta donde se ha realizado la instalación. Lo más normal es que sea la siguiente "C:\Archivos de Programa\OpenCV\bin\". La ruta puede variar según la configuración del equipo, el sistema operativo o dependiendo de la instalación de OpenCV (puede no estar en C: o en Archivos de Programa). Para instalar las librerías los ejecutables o las fuentes se pueden encontrar tanto en *http://opencv.willowgarage.com/wiki/* como en *http://sourceforge.net/projects/opencvlibrary/*. Aquí se pueden encontrar tanto para Windows, como para Linux.

3.1.2. Configurar opciones globales

Una vez incluida la dirección de librerías al PATH, se deben configurar las opciones globales. Para ello se inicia Visual Studio C++ y se selecciona en el menú **Herramientas → Opciones**. Después se selecciona **Proyectos y Soluciones → Directorios de VC++**. En el listbox con nombre "Mostrar directorios para:" de arriba a la derecha se debe seleccionar **Archivos de**

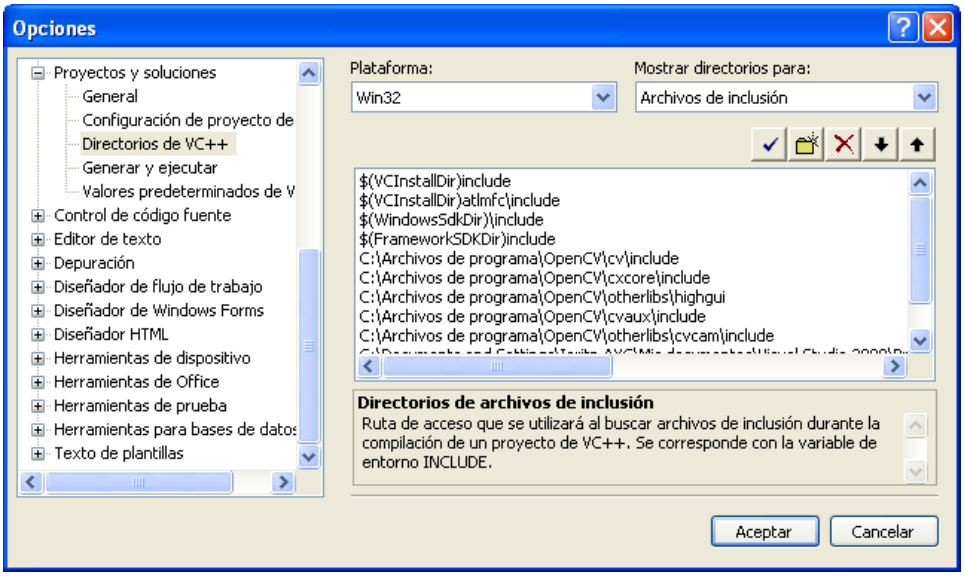
biblioteca. Una vez hecho esto se deben añadir las siguientes líneas:

"C:\Archivos de programa\OpenCV\lib"
"C:\Archivos de programa\OpenCV\bin"



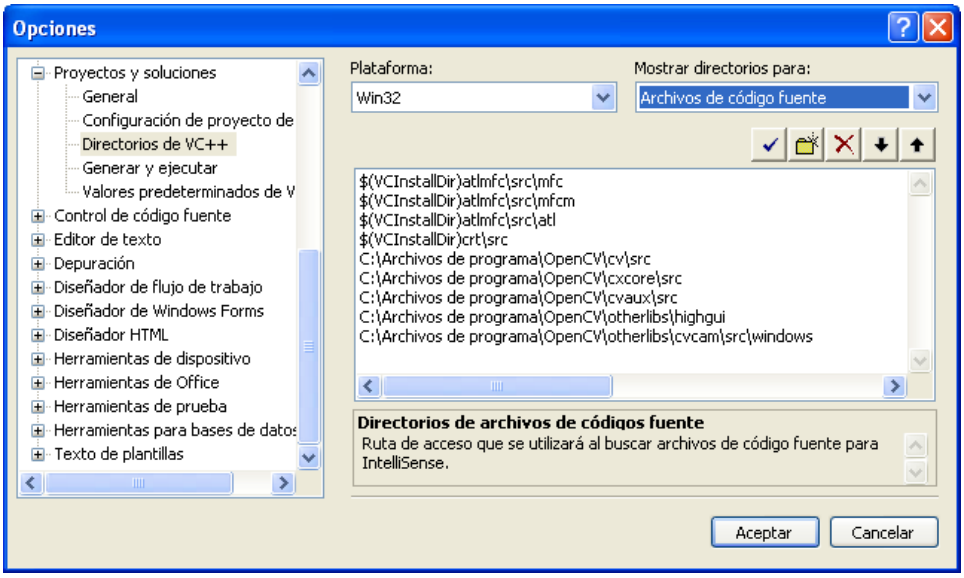
Después se debe seleccionar en el listbox **Archivos de inclusión** y se deben añadir las siguientes líneas:

"C:\Archivos de programa\OpenCV\cv\include"
"C:\Archivos de programa\OpenCV\cxcore\include"
"C:\Archivos de programa\OpenCV\otherlibs\highgui"
"C:\Archivos de programa\OpenCV\cvaux\include"
"C:\Archivos de programa\OpenCV\cvcam\include"



Después se debe seleccionar en el listbox **Archivos de código fuente** y se deben añadir las siguientes líneas:

"C:\Archivos de programa\OpenCV\cv\src"
"C:\Archivos de programa\OpenCV\cxcore\src"
"C:\Archivos de programa\OpenCV\cvaux\src"
"C:\Archivos de programa\OpenCV\otherlibs\highgui"
"C:\Archivos de programa\OpenCV\otherlibs\cvcam\src\windows"

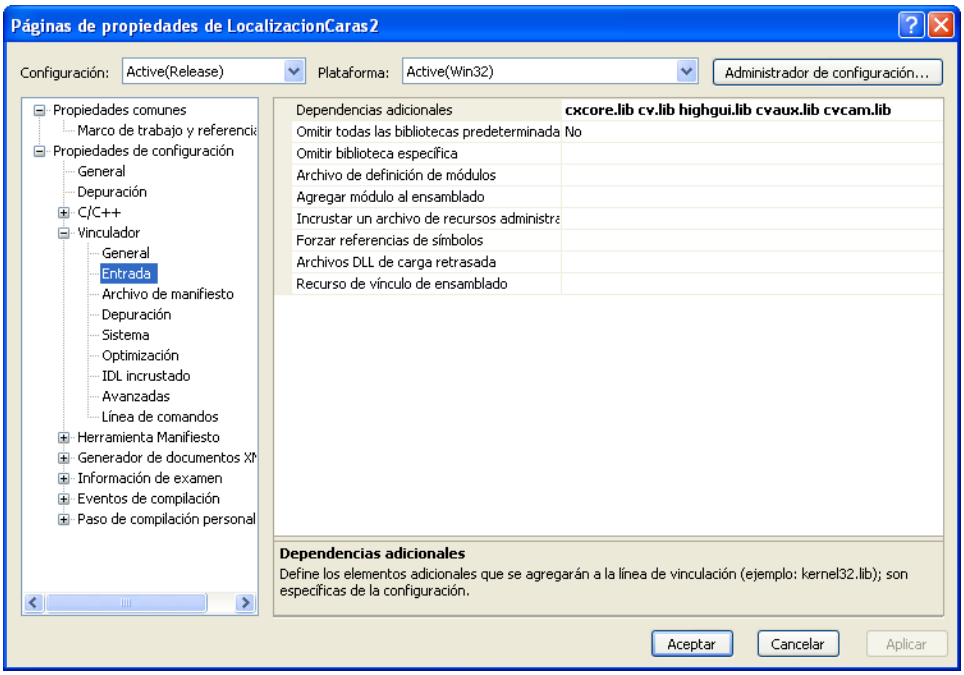


Este apartado solo debe realizarse una vez. Forma parte de la configuración de Visual Studio C++.

3.1.3. Crear nuevo proyecto

Para crear un proyecto nuevo se debe seleccionar **Archivo → Nuevo → Proyecto**. En el apartado **Tipos de proyecto** se debe seleccionar **Visual C++ → Win32** y una vez ahí se puede seleccionar **Aplicación de consola win32** o **Proyecto win32**. Una vez hecho esto, se ejecutará un asistente en el que se debe hacer clic en finalizar. Después se abre el proyecto creado y se selecciona **Proyecto → Propiedades de [nombre de proyecto]**. Aquí se selecciona **Propiedades de configuración → Vinculador → Entrada** y en el apartado **Dependencias adicionales** se debe incluir la siguiente línea **"cxcore.lib cv.lib highgui.lib cvaux.lib cvcam.lib"**. Falta decir que cada vez que se quiera utilizar alguna función o estructura de OpenCV en un fichero fuente hará falta escribir las siguientes líneas en el fichero del código:

#include <cv.h>
#include <cxcore.h>
#include <highgui.h>



3.2. Uso de la Librería OpenCV

3.2.1. OpenCV

OpenCV son un conjunto de librerías, bajo licencia BSD, desarrolladas en C y C++, que comprenden una gran cantidad de algoritmos de visión por computador. Se están desarrollando interfaces para poder trabajar con OpenCv en Python, Ruby, Matlab y otros lenguajes. OpenCV fue diseñado con el propósito de obtener una buena eficiencia computacional y poder realizar aplicaciones en tiempo real. El código está escrito en C optimizado y puede obtener buen rendimiento de procesadores con múltiples núcleos. Al ser Intel una de las compañías que tomó parte en su desarrollo, se puede obtener optimización automática en arquitecturas de esta compañía utilizando IPP (Intel Performance Primitives). Las librerías de OpenCV están organizadas de la siguiente manera:

- **CxCore**: Contiene funciones para estructuras de datos, algebra de matrices, transformación de datos, persistencia de objetos, manejo de memoria, manejo de errores, carga dinámica de código, dibujo, texto y matemática básica.
- **CvReference**: Contiene funciones para procesamiento de imágenes, análisis de estructura de imágenes, captura de movimiento, reconocimiento de patrones y calibración de cámaras.
- **CvAux**: Contiene interfaces de usuario y funciones de imagen/video y memoria.
- **HighGui**: Contiene funciones para correspondencia estéreo, cambio de forma del punto de vista de las cámaras, seguimiento 3D en estéreo, funciones de reconocimiento de objetos por eigenvalores (eigenfaces y PCA), modelos de ocultos de Markov embebidos, descriptores de texturas, segmentación objetofondo, etc.
- **Machine Learning**: Contiene muchas funciones de clustering, clasificación y de análisis de datos.
- **CvCam**: Esta sección es muy parecida a la sección Highgui. Tiene las mismas funcionalidades definidas de otra forma. CVcam ya no se suele encontrar en las últimas versiones de OpenCV, si se quiere utilizar es necesario descargarse una versión antigua de OpenCV o descargarse el fichero que contiene esta sección por separado e integrarlo en el entorno de desarrollo.

Existen multitud de recursos para obtener información sobre cualquiera de las funciones de OpenCV, estos recursos van desde libros, manuales y sobre todo Internet. Hay varias páginas en las que se puede consultar el comportamiento de las funciones de OpenCV, las que más se han utilizado en este proyecto son *http://opencv.willowgarage.com/documentation/index.html* y *http://www.cs.indiana.edu/cgi-pub/oleykin/website/OpenCVHelp/*, todo lo que no se encuentra aquí se ha consultado en los numerosos foros, listas de correo o blogs sobre OpenCV que hay en la red.

3.2.2. Particularidades con OpenCV en este proyecto

La estructura para tratamiento de imágenes en OpenCV es `IplImage`, que tiene varios canales. Concretamente tiene uno si la imagen es en blanco y negro y tres si es en color. Para obtener los canales de la imagen en color para poder tratarlos por separado y para poder juntarlos después para crear la imagen nueva hay dos funciones, `cvSplit` y `cvMerge` respectivamente. Estas funciones consumen mucho tiempo de ejecución por lo que se ha construido una estructura imagen que consta de tres `IplImage` para crear la imagen a color y así no hace falta ir juntando y separando cada vez los planos con el gasto de recursos del sistema que esto supone. El siguiente código muestra la estructura de la clase imagen y algunas de las funciones que tiene.

```
class Imagen{
IplImage *canal1;// B - H
IplImage *canal2;// G - S
IplImage *canal3;// R - V
IplImage *canal4;// auxiliar1
IplImage *canal5;// auxiliar2
IplImage *canal6;// auxiliar3

public:
Imagen(CvSize dimension);
Imagen(IplImage *imagen);
~Imagen();
int acceder(int canal, int i, int j);
void modificar(int canal, int i, int j, int num);
void modificar(int i, int j, int num);
void clonar(IplImage *imagen);
CvSize dimension();
...
```

3.3. Problemas y soluciones

Los siguientes, son los problemas que se han encontrado a lo largo del proyecto, así como las soluciones que se les han dado a cada uno de ellos.

3.3.1. Cámaras

El primer problema es el de las cámaras. Al principio el proyecto se había planteado para realizarlo con unas cámaras del tipo de las que se usa en sistemas de vigilancia. Estas cámaras se conectan a una tarjeta capturadora de video que va conectada al ordenador. La tarjeta capturadora es una tarjeta de la marca Matrox y funciona con MIL (Matrox Imaging Library), librería donde se encuentran los controladores para que OpenCV pueda identificar la cámara. En las páginas web oficiales de OpenCV se indica que las cámaras que se quieran identificar a través de OpenCV en el sistema operativo Microsoft Windows, deben conectarse mediante uno de los siguientes drivers VFW (Video for Windows) o MIL. Como ya se ha mencionado anteriormente en este proyecto se ha usado MIL y no parece buena idea utilizar VFW ya que es un driver muy antiguo, la última versión encontrada era de 1994. El poblema es que OpenCV no se comunica con la cámara aunque se utilice el driver que ellos recomiendan. La solución a este problema ha sido la utilización de cámaras web que OpenCV identifica automáticamente.

3.3.2. Problemas con la documentación oficial de OpenCV

El segundo problema se ha encontrado una vez se ha empezado a programar. La documentación sobre las funciones de OpenCV encontrada en sus páginas oficiales en ocasiones es insuficiente. Hay varias funciones que no queda claro cómo utilizarlas o qué parámetros hay que pasarle. La solución a este problema es consultar foros, blogs y listas de correo. Hay cantidad de ellos y son de gran utilidad.

3.3.3. Punteros

Los problemas con los punteros se pueden dividir en dos y en ambos casos tienen que ver con OpenCV.

El primer problema se puede observar al empezar a trabajar con OpenCV y sus objetos y funciones. La mejor forma de ilustrarlo es mediante un ejemplo. Si se quiere trabajar con una imagen en OpenCV primero se declara, luego se inicializa la estructura y luego se asignan los valores. Por ejemplo:

```
IplImage *imagen;
CvCapture* capture= cvCaptureFromCAM(-1);

imagen = cvCreateImage( tamaño, nBits, canales );
imagen = cvQueryFrame(capture);
```

Esta forma de programar es extraña. Capture es un capturador de imágenes, al ejecutar la función `cvQueryFrame` sobre él devuelve una imagen, por lo tanto, no debería hacer falta inicializar la imagen con la función `cvCreateImage`, ya que la imagen ya está creada. Pero si no se realiza de esta forma el resto del programa que dependa de esa imagen, que seguramente sea todo el resto del programa, se comporta de forma errónea. Esta es una de las razones por las que se afirma que los punteros en OpenCV se comportan de forma extraña, no habitual en otras librerías o lenguajes o en la misma API de C.

Hay otro ejemplo si se sigue con la programación del ejemplo anterior. Al estar programando en C++, cada objeto que se crea debe ser destruido cuando deja de ser útil o consumirá memoria que no hace falta consumir y que puede ser valiosa para otros fines o para no acabar colapsando la memoria. Así pues, hay que llamar al destructor, para ello se seguiría programando de la siguiente forma:

```
(... uso de la imagen)

cvReleaseImage(&imagen);
```

La forma de pasarle el objeto a la función `cvReleaseImage` es extraña ya que en la mayoría de APIs se le pasaría sin `&` pero aquí es necesario. Aunque esto no da muchos problemas, ya que en todos los ejemplos de OpenCV que se encuentran se deja claro, no deja de ser un comportamiento extraño. No solo ocurre con la función para liberar imagenes, sino que ocurre con todas las funciones para liberar objetos, como por ejemplo `cvReleaseCapture` o `cvReleaseHaarClassifierCascade` entre otras.

A continuación se detalla otro problema con punteros con OpenCV. Este es el mayor problema que se ha tenido, el que más ha costado localizar y resolver en todo el proyecto. A la hora de trabajar con la estructura IplImage, en mitad de la ejecución las imágenes se alteran y dejan de mostrar lo que deben mostrar y muestran una imagen por defecto, como si se mostrase una imagen sin inicializar. El problema es que el puntero que apunta a la imagen se modifica en mitad de la ejecución y la imagen cambia y al estar sin inicializar, muestra la de por defecto.

Para solucionar esto se declara un doble puntero. Se pasa a todas las

funciones como doble puntero y cuando hace falta realizar operaciones sobre la imagen se le aplica `&`. Es una solución poco elegante, pero como más adelante se decide trabajar con la estructura imagen de creación propia y no con la estructura IplImage de la librería OpenCV, ni se da el problema ni se tiene la necesidad de utilizar esta solución.

3.3.4. Umbrales originales

Un problema importante que se ha encontrado es que pese a que los umbrales que se han obtenido mediante el estudio en el artículo, estos no se adaptan a las condiciones de luminosidad del laboratorio. Al realizar pruebas con los umbrales del articulo se dan demasiados falsos positivos y falsos negativos. Esto se debe a que el laboratorio sufre unas condiciones de luminosidad muy elevada y las imágenes son demasiado claras, lo que hace que unos umbrales obtenidos en condiciones de luminosidad normal no sirvan. Por este motivo los umbrales han sido ajustados para que se adecuen mejor a las condiciones de luz del laboratorio. De esta forma se obtienen mejores resultados ya que se disminuye el número de falsos negativos y de falsos positivos.

Al final los umbrales quedan de esta manera.

$T_{H1} = 10^{o} \leq H \leq T_{H2} = 34^{o}$

$S_{S1} = 58{,}65^{o} \leq S \leq S_{S2} = 255^{o}$

Para ajustar los umbrales se han separado los tres canales de la imagen HSV y se ha observado por separado bajo las condiciones de luminosidad del laboratorio, cuál de los canales tiene información para poder diferenciar el color de la piel. Al realizar esta prueba se ve claramente que los canales H y S tienen información muy valiosa ya que se diferencian las partes de color piel de las que no lo tienen, pero en el canal V no se diferencia la piel. Por este motivo no tiene sentido fijar umbrales, ya que este canal no guarda información para poder realizar la diferenciación.

Los valores iniciales para los umbrales de los canales H y S se han intuido mediante las pruebas. Posteriormente, realizando más pruebas para ello, se han ido ajustando esos valores iniciales haciendo que disminuya el número de falsos negativos y de falsos positivos.

3.3.5. Codec

El siguiente, es uno de los problemas que se tuvo con la documentación de OpenCV. La función de OpenCV de captura de video a través de fichero

`cvCaptureFromFile`, a la que se le pasa como argumento el nombre del fichero de video que se quiere utilizar o la ruta completa si el video no está en el directorio en el que trabaja la aplicación, no es capaz de capturar de cualquier tipo de video. El video debe estar en formato avi y codificado con el codec *Microsoft Video 1* o no será capaz de capturar ninguna secuencia de ese fichero. El problema es que este inconveniente no se menciona en la documentación de OpenCV, pero como se ha comentado anteriormente, se puede encontrar realizando una búsqueda en Internet a través de los diversos foros y blogs sobre OpenCV que existen en la red.

3.3.6. Velocidad

A la hora de realizar pruebas, cuando se ejecuta el programa es necesario que una persona pase por delante de la cámara de vídeo mientras el programa se está ejecutando. Realizar y observar la prueba a la vez, aunque posible no es nada cómodo y no se llegan a observar bien los resultados, sobre todo al principio y al final del vídeo. Otra posibilidad es la de disponer de una persona para pasar por delante de la cámara, pero no es posible tener una persona para ejecutar todas las pruebas requeridas.

Dado este problema, se ha pensado en realizar varias grabaciones de vídeo en ficheros, con los que se pueden ejecutar pruebas sobre los ficheros. El problema es que a la aplicación le cuesta mucho tiempo analizar las imágenes desde el fichero vídeo. Al costarle tanto analizar cada imagen, no da sensación de movilidad porque en la siguiente imagen no se da un cambio de posición muy grande. Por este motivo, se ha decidido no analizar todas las imagenes, se puede elegir cada cuantas imagenes se quiere realizar el análisis y mostrar el resultado. Este problema no es tan grave cuando la captura de imágenes se realiza desde la cámara de vídeo.

Capítulo 4

RESULTADOS EXPERIMENTALES

4.1. Explicación del experimento

Para realizar las pruebas se debe establecer un procedimiento de pruebas con el que se pueda observar la eficacia de la aplicación. Para ello se deben crear situaciones para que pueda suceder que se den tanto aciertos negativos como positivos y falsos positivos como negativos.

El experimento que se establece consiste en lo siguiente. Se inicia la ejecución de la aplicación no habiendo ninguna persona delante de la cámara. Después de iniciar la aplicación una persona se aproxima hacia el área de grabación de la cámara con la cara de costado. Posteriormente gira la cara para mostrarla a la cámara, luego vuelve a girarla para no mirar directamente a la cámara y se sale del área de grabación.

En la ejecución se observa si al principio la aplicación detecta alguna cara, que sería un falso positivo. Al principio es trivial ya que no hay ninguna persona delante del área de grabación pero una vez que aparece la persona en el área de grabación, no debe mostrar la cara (acierto negativo) hasta que la persona empiece a girarse y a mostrar la cara por completo, de lo contrario se produce un falso positivo. Una vez que la persona aparece con la cara mirando hacia la cámara debe mostrar la cara (acierto positivo), de lo contrario se produce un falso negativo. Después, una vez vuelve a girar la cara la aplicación debe dejar de mostrar la cara (acierto negativo) de nuevo, de lo contrario sería un falso positivo. Como se puede observar el experimento está pensado para que puedan darse los cuatro casos.

Este proceso se repetirá varias veces con personas diferentes para probar

que funciona para varios individuos y no para uno solo.

4.2. Resultados

Estas son varias de las pruebas realizadas en el laboratorio con el fin de probar la aplicación. Como se puede observar se muestran las fases que se han descrito anteriormente en el apartado de explicación del experimento.

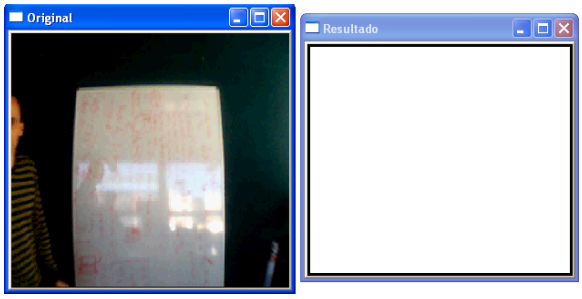


Figura 4.1: Fase en la que no localiza la cara.

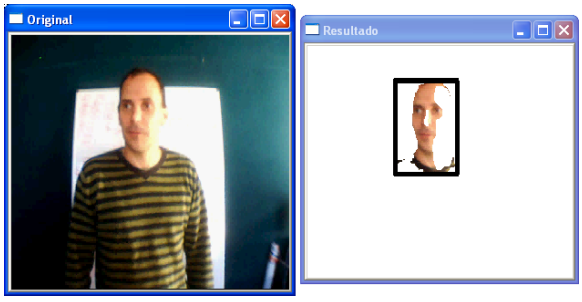


Figura 4.2: Fase en la que localiza la cara.

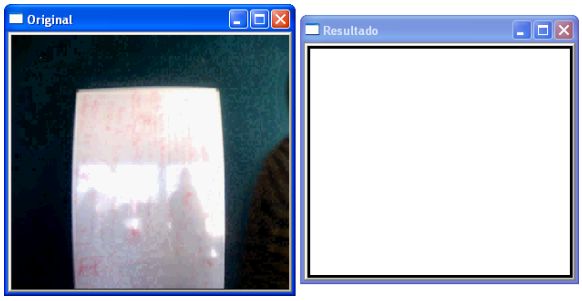


Figura 4.3: Fase en la que deja de localizar la cara.

En ejemplo de la figuras 4.1, 4.2 y 4.3, en la fase en la que localiza la cara se puede observar la importancia de la iluminación. Hay gran cantidad de

luz que proviene de la parte derecha que hace que la cara se muestre blanca y que la aplicación no reconozca esa parte de la cara. Por este motivo se ha hecho necesario el tener que ajustar los umbrales de los canales HSV como ya se ha explicado anteriormente.

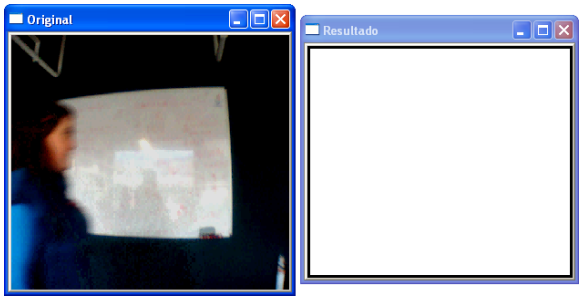


Figura 4.4: Fase en la que no localiza la cara.

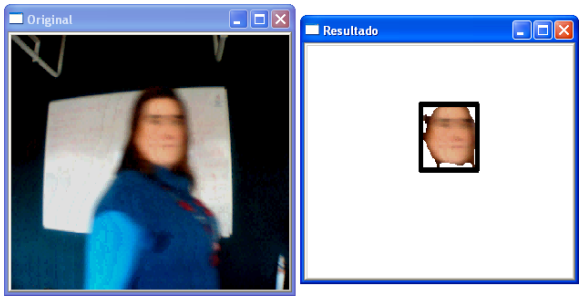


Figura 4.5: Fase en la que localiza la cara.

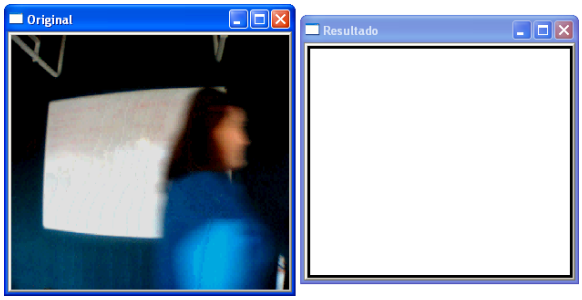


Figura 4.6: Fase en la que deja de localizar la cara.

En las figuras 4.4, 4.5 y 4.6, se observa que la aplicación funciona bien para estos casos. Durante la ejecución de la aplicación puede saltar algún falso positivo, pero son muy escasos y no tienen gran importancia.

Con el vídeo de las imágenes de las figuras 4.7, 4.8 y4.9, la aplicación se comporta correctamente en la mayoría de los fotogramas pero, hay un

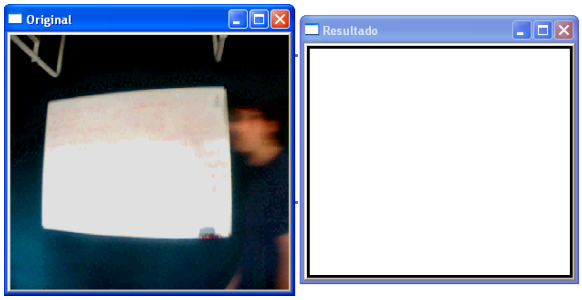


Figura 4.7: Fase en la que no localiza la cara.

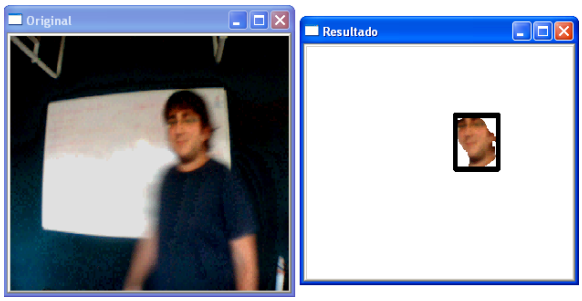


Figura 4.8: Fase en la que localiza la cara.

fragmento del vídeo en el que se producen falsos negativos estando la cara claramente mirando a la cámara de frente. Este hecho se puede observar en la figura 4.10.

En el vídeo de las imágenes de las figuras 4.11, 4.12 y 4.13, la aplicación se comporta correctamente salvo en ciertos fotogramas, en las que la aplicación reconoce como cara el brazo de la persona que pasa por delante de la cámara, por lo que se produce un falso positivo. Este hecho se puede observar en la figura 4.14.

En el vídeo de las figuras 4.15, 4.16 y 4.17 la aplicación no se comporta correctamente ya que no localiza la cara del todo bien. En la mayoría de los fotogramas no localiza la cara. En algunos fotogramas se puede observar cómo si localiza la cara pero no se puede visualizar. Este comportamiento es debido a que la persona que pasa por delante de la cámara tiene barba y su cara ya no tiene un color carne sino que es más oscuro. Es un comportamiento normal ya que la aplicación está implementada para filtrar sólo los píxeles de color carne.

En la tabla 4.1 se puede observar la tasa de aciertos positivos, aciertos negativos, falsos positivos y falsos negativos que tienen cada uno de los vídeos. Se puede observar que como ya se ha comentado anteriormente, con

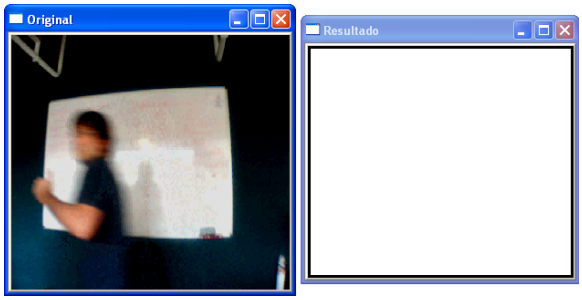


Figura 4.9: Fase en la que deja de localizar la cara.

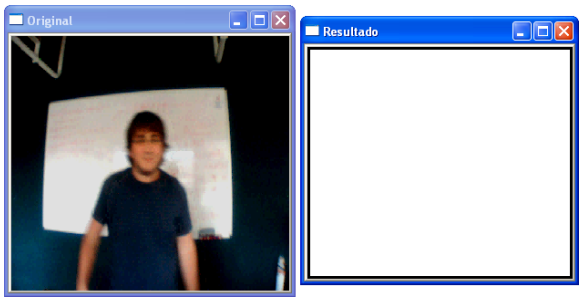


Figura 4.10: Falso negativo

los dos primeros vídeos el programa tiene un comportamiento muy bueno, con los dos siguientes bueno y con el último bastante malo. El motivo de este comportamiento ya se ha explicado anteriormente.

Vídeo\Tasa	Acierto Positivo	Acierto Negativo	Falso Positivo	Falso Negativo
Figuras 4.1 - 4.3	Elevada	Elevada	Baja	Baja
Figuras 4.4 - 4.6	Elevada	Elevada	Baja	Baja
Figuras 4.7 - 4.10	Elevada	Normal	Baja	Normal
Figuras 4.11 - 4.14	Normal	Elevada	Normal	Baja
Figuras 4.15 - 4.17	Baja	Elevada	Elevada	Baja

Cuadro 4.1: Tabla de tasas de comportamiento.

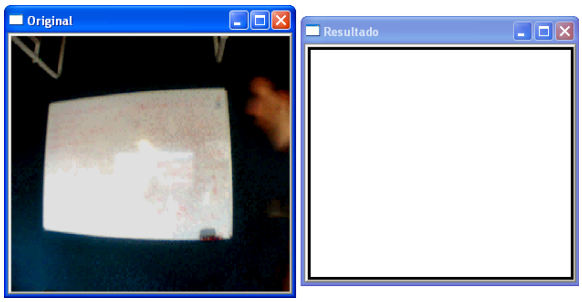


Figura 4.11: Fase en la que no localiza la cara.

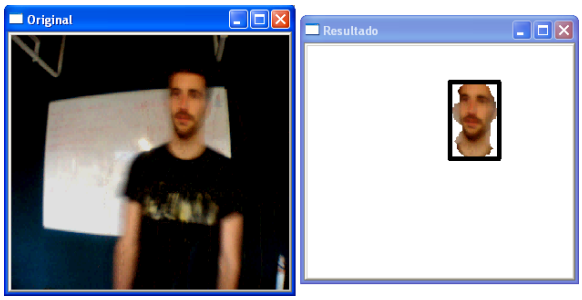


Figura 4.12: Fase en la que localiza la cara.

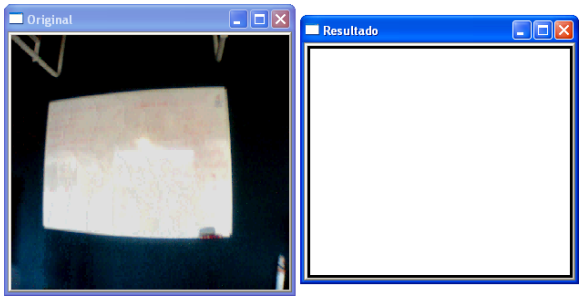


Figura 4.13: Fase en la que deja de localizar la cara.

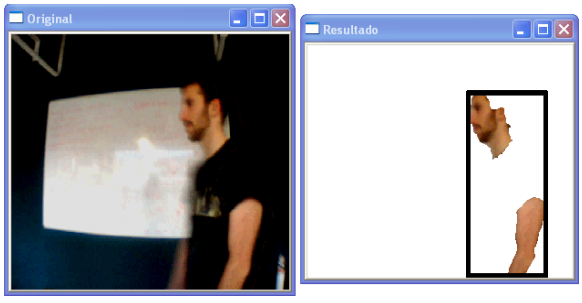


Figura 4.14: Falso positivo

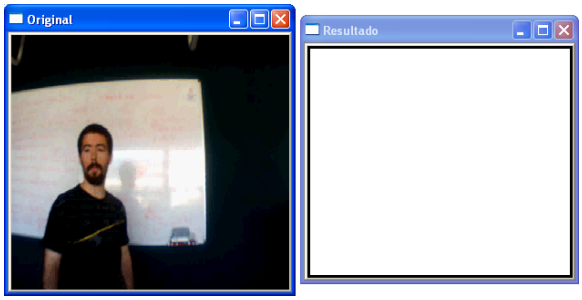


Figura 4.15: Fase en la que no localiza la cara.

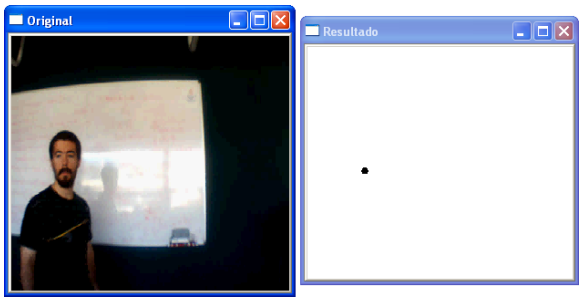


Figura 4.16: Uno de los fotogramas en los que localiza la cara.

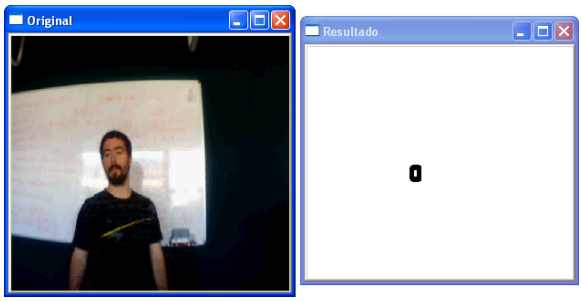


Figura 4.17: Otro de los fotogramas en los que localiza la cara.

Capítulo 5

UNIÓN DE PROYECTOS

Uno de los propósitos que han surgido a lo largo de la realización de este proyecto es el de realizar una fusión con otro proyecto. El otro proyecto trata también de localizar caras pero lo realiza con una técnica diferente. Se pretende que al unir los dos proyectos se puedan descartar varios falsos positivos, y por tanto, ganar precisión.

El otro proyecto trata de localizar caras mediante la medición de las distancias que hay entre los ojos, nariz y boca. Estas medidas guardan una proporción y se puede definir un rango entre el que se deberían hallar para poder decir que se trata de una cara.

5.1. Unión de proyectos

Una vez que los dos proyectos funcionan bien por separado, se procede de la siguiente forma. Se decide que primero se ejecute el localizador de caras de este proyecto, ya que con este proyecto se elimina mucho ruido y manchas que por su forma o color no pueden ser caras. Y después ejecutar el localizador de caras basado en medir distancias biométricas para eliminar los falsos positivos que el primer localizador encuentra. De realizarlo en orden inverso, el localizador de medidas biométricas se ejecutaría demasiadas veces en vano, ya que debería buscar por toda la imagen ojos, nariz y boca por toda la imagen. Teniendo en cuenta que suelen ser manchas negras y que las imágenes suelen tener mucho ruido, sería bastante probable que se ejecutase muchas veces sobre manchas negras que no pertenecen a ojos, nariz y boca.

Una vez seleccionado el orden en el que se ejecutan los dos localizadores, se decide ejecutar el segundo localizador en un hilo. De esta forma se pueden ejecutar los dos localizadores simultáneamente. Cuando el primer localizador

le pasa al segundo las áreas de la imagen sobre las que la trabajar, el segundo localizador se ejecuta y mientras el primero vuelve a empezar con el siguiente fotograma para finalizarlo cuanto antes y poder pasárselo al segundo cuando acabe con el anterior fotograma. De esta forma se intenta obtener un mejor rendimiento del programa y poder obtener una ejecución lo más cerca posible a tiempo real.

El esquema a seguir sería el que se muestra en la figura 5.1.

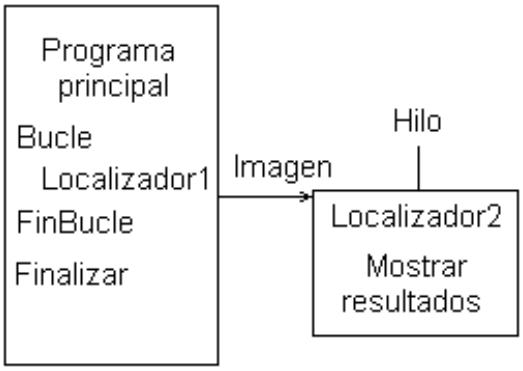


Figura 5.1: Esquema de la union de programas.

5.2. Programar hilos en C++

Como se ha indicado en la sección donde se dice como unir los proyectos, se utilizan hilos. Hay numerosas formas de programar con hilos en C++. En este apartado se va a explicar la que se ha utilizado en este proyecto.

En este proyecto se han implementado los hilos mediante la clase `pthread`. El primer paso es instalar y configurar esta biblioteca para que funcione en Visual Studio C++. Los pasos a seguir son los siguientes. Primero hay que descargar la biblioteca `pthread` para Windows, se puede descargar de *http://sourceware.org/pthreads-win32/*. Una vez descargado, en vez de ejecutarlo se descomprime. Dentro del directorio llamado Pre-built hay otros dos directorios, uno con los archivos de cabecera llamado \include y otro con las librerías para el enlazador o linker llamado \lib. El siguiente paso consiste en copiar los 3 archivos .h que estan en el directorio include (pthread.d, semaphore.h, sched.h), dentro del directorio /include del Visual C++ y en copiar el archivo de libreria pthreadVC1.lib que esta dentro del directorio /lib, al directorio /lib del Visual C++. Después se debe copiar el archivo DLL pthreadVC1.dll que también esta dentro del directorio /lib al directorio "C:\Windows\system32 ". Por último, lo único que falta es poner en

propiedades de tu proyecto, **Proyecto → Propiedades de [nombre de proyecto]**. Aquí se selecciona **Propiedades de configuración → Vinculador → Entrada** y se debe poner pthreadVC1.lib.

Para programar con `pthread` se debe implementar un método llamado `MyStub` en la clase que se quiere que se comporte como un hilo. La cabecera del método debe tener la siguiente estructura `void* clase_hilo::MyStub (void* arg)` y se debe llamar desde la clase principal de la siguiente forma.

```
pthread_t threads[1];
rc = pthread_create(&threads[0], NULL, &clase_hilo::MyStub, (void
*)rz);
```

Donde `rz` es un objeto de la clase `clase_hilo`.

5.3. Resultados

En esta sección se muestran los resultados obtenidos con la aplicación basada en la unión de los dos proyectos.

En las figuras 5.2 y 5.6 los resultados no son los esperados. En las figura 5.3 los ojos y la nariz se localizan bien pero no es así con la boca. En la figura y 5.4 la nariz se localiza bien pero no es así con los ojos y la boca. Y por último en las figuras 5.5 y 5.7 los ojos nariz y boca se localizan mejor.

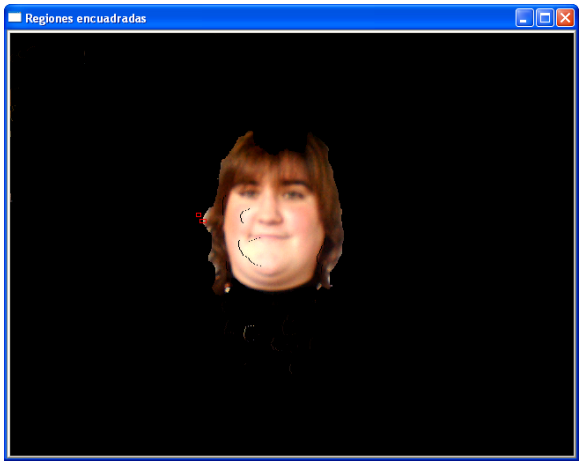


Figura 5.2: No localiza bien las distancias biométricas.

Aunque los ojos, cara y boca no se localicen prefectamente, el sistema funciona bien ya que las caras se localizan. También se ha obtenido una mejora ya que antes se daban falsos positivos, como puede ser un brazo, y ahora no se dan tanto. Si un brazo sale los cuadrados localizadores de ojos,

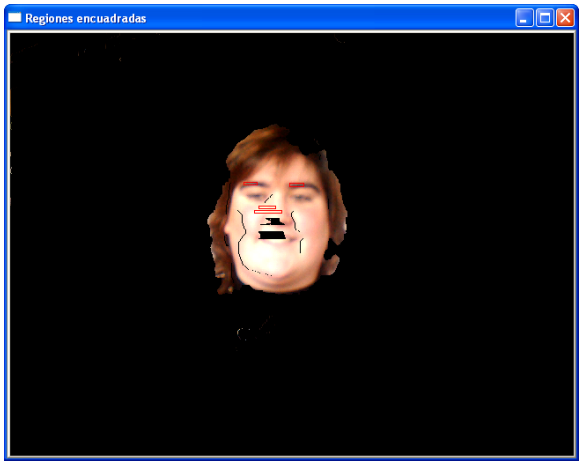


Figura 5.3: Localiza bien las distancias biométricas excepto la boca.

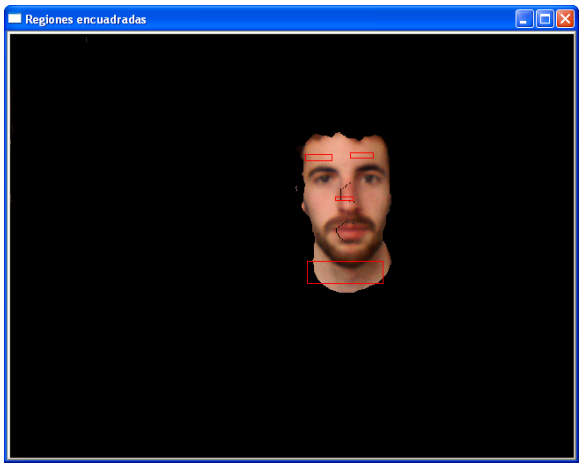


Figura 5.4: Localiza bien las distancias biométricas excepto la boca.

nariz y boca se dibujan encima de la posición de la cara en vez de encima de la posición del brazo.

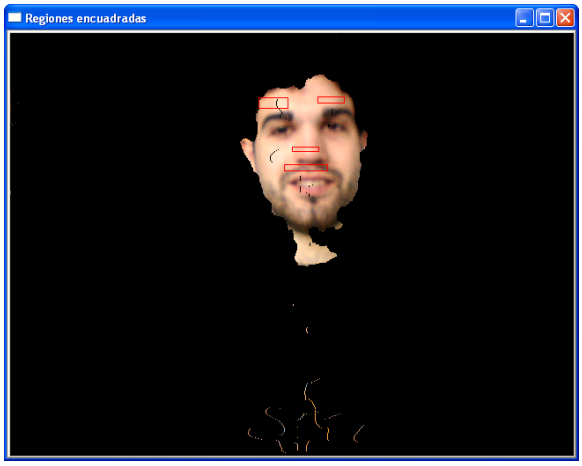


Figura 5.5: Localiza bien las distancias biométricas.

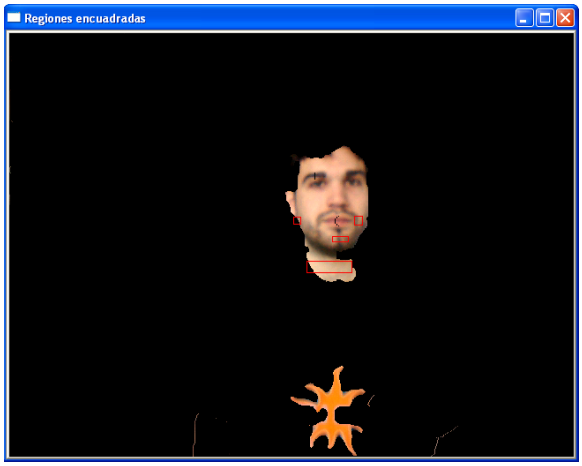


Figura 5.6: No localiza bien las distancias biométricas.

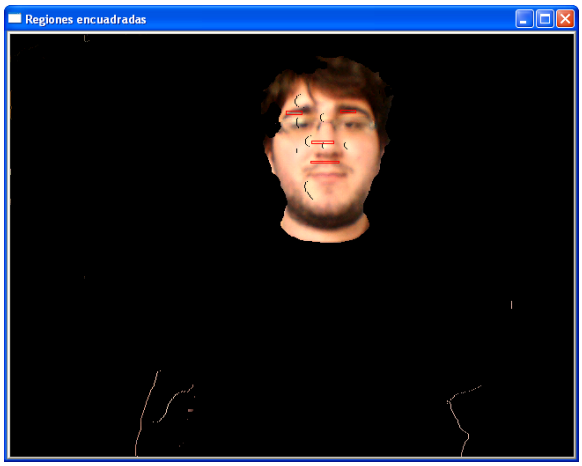


Figura 5.7: Localiza bien las distancias biométricas.

Capítulo 6

CONCLUSIONES Y LÍNEAS FUTURAS

6.1. Conclusiones

A pesar de las dificultades que ha supuesto realizar el proyecto y terminar de implementarlo, los resultados obtendidos a la hora de localizar caras son bastante satisfactorios. El programa localiza la mayoría de las caras si se hace en el laboratorio durante las horas de luz natural. Se debe tener en cuenta que la implementación realizada solo funciona bajo condiciones lumínicas y de entorno controladas. Aunque esto parezca una restricción un poco fuerte es bastante normal. Un programa capaz de reconocer y localizar una cara en cualquier condición lumínica y de entorno sería extremadamente complejo y pesado a la hora de ejecutar.

En cuanto a la unión de proyectos, el resultado también es satisfactorio, aunque no de unos resultados tan buenos como un localizador por separado. Es normal, ya que no se ha invertido tanto tiempo a la hora de calibrar los dos sistemas trabajando simultáneamente. No obstante las caras se localizan y no se producen falsos positivos. No se evitan tanto los falsos negativos.

6.2. Líneas Futuras

A pesar de que el proyecto ha obtenido resultados satisfactorios, hay una gran cantidad de cosas en las que puede mejorar y mucho. En esta sección se proponen una serie de posibles líneas futuras para seguir con el proyecto y mejorarlo. Por supuesto no se proponen todas las posibles.

Una de las líneas futuras a seguir más importantes sería la optimización del programa. Aunque no es demasiado lento, no se consigue que se ejecute en tiempo real, una característica bastante deseable en este tipo de aplicaciones. Para ello habría que buscar en el programa partes del código que se puediesesn optimizar en cuanto a coste de ejecución. También se podría intentar conseguir no ejecutar el algoritmo para todas las secuencias en intentar implementar un algoritmo de seguimiento de la cara. Para que esta solución de resultado este algoritmo debería ser más eficiente que el de localización.

Aunque en el apartado de conclusiones se ha mencionado que no es fácil realizar una aplicacion de estas características capaz de adaptarse a cualquier condición lumínica, una de las posibles líneas futuras podría ser conseguir una mayor flexibilidad ante un cambio de esta condición. Se podría implementar la característica analizando previamente la cantidad de luminosidad de la imagen y adaptando los umbrales de color en función de esta. Cuanta mayor flexibilidad se quiera conseguir más complejo sería este apartado.

Otra de las posibles líneas futuras podría ser intentar reducir la pérdida de buen comportamiento que sufre la aplicación con las personas con barba. Este tipo de aplicaciones no pueden salir adelante si el hecho de que un individuo tenga barba impide su correcto funcionamiento.

Para finalizar se propone una línea futura más. Esta línea es la única que no pretende mejorar el comportamiento de la aplicación sino que pretende añadirle una característica más, el reconocimiento de caras. Una vez que se tiene la cara localizada, el siguiente paso lógico a seguir sería intentar reconocerla, es decir intentar asociar la cara con la identidad del individuo de la misma. Para ello habría que implementar un algoritmo de reconocimiento de caras con alguna de varias de las técnicas que existen con ese propósito. Para poder combinar esta línea futura con las aneriormente mencionadas, se debería intentar implementar una solucion eficaz y eficiente, ya que sino se volvería a perder la característica de que la aplicación funcione en tiempo real.

Apéndice A

MANUAL DE OPENCV

A.1. Manual

Este apartado es un manual de las funciones de OpenCV que se han ido nombrando a lo largo de este documento. En él se detallan los argumentos de las funciones y las opciones de los mismos. Se puede obtener esta y más información en *http://opencv.willowgarage.com/documentation/*.

`void cvCvtColor(const CvArr* src, CvArr* dst, int code)`

Convierte una imagen de un espacio de color a otro.

Parámetros:

- `src`: imagen original de 8, 16 o 32 bits.
- `dst`: imagen destino del mismo número de bits. El número de canales puede cambiar.
- `code`: código que indica la conversión. En el caso de rgb a hsv `CV_BGR2HSV` otros ejemplos son `CV_BGR2XYZ`, `CV_BGR2YCrCb` o `CV_BGR2HLS`.

`void cvSmooth(const CvArr* src, CvArr* dst, int smoothtype=CV_GAUSSIAN, int param1=3, int param2=0, double param3=0, double param4=0)`

Realiza filtrados sobre una imagen.

Parámetros:

- `src`: imagen original de 8, 16 o 32 bits.
- `dst`: imagen destino del mismo número de bits. El número de canales puede cambiar.

- `smoothtype`: tipo de filtro:
  - `CV_BLUR_NO_SCALE`: filtro lineal en el que se trata de hacer una convolución con `param1 x param2`.
  - `CV_BLUR`: filtro lineal en el que se trata de hacer una convolución con `param1 x param2` con subsecuente escalado con $1/(param1 \cdot param2)$.
  - `CV_GAUSSIAN`: filtro lineal en el que se trata de hacer una convolución con `param1 x param2` con kernel gaussiano.
  - `CV_MEDIAN`: filtro de la mediana con ventana `param1 x param1`.
  - `CV_BILATERAL`: filtro bilateral con ventana `param1 x param1` sigma de color igual a `param3` y sigma espacial igual a `param4`. Si `param1` es 0 la ventana es cuadrada de dimensión `cvRound(param4 * 1.5) * 2 + 1`.
- `param1`: dimensión de anchura. Debe ser positivo e impar.
- `param2`: dimensión de altura. Ignorado por los métodos `CV_MEDIAN` y `CV_BILATERAL`. En caso de no ser ignorado y ser igual a 0 se iguala a `param1`. Si se indica debe ser positivo e impar.
- `param3`: en el caso de un filtro Gaussiano, este parámetro indica la desviación estándar. Si es igual a 0, se calcula en función del tamaño del kernel. $\sigma = 0.3$(n / 2 - 1) + 8 donde n = `param1` para el kernel horizontal y `param2` para el kernel vertical.
- `param4`: únicamente se usa en el caso del filtro bilateral y está explicado en ese apartado.

`IplImage`

IplImage es una estructura con los siguientes campos.

- `int nSize`: el resultado de `sizeof(imagen)`.
- `int ID`: versión siempre igual a 0.
- `int nChannels`: número de canales, generalmente de 1 a 4.
- `int alphaChannel`: ignorado por OpenCV.
- `int depth`: número de bits con el que se guarda la imagen. Los valores admitidos son:
  - `IPL_DEPTH_8U`: enteros de 8 bits sin signo.
  - `IPL_DEPTH_8S`: enteros de 8 bits con signo.
  - `IPL_DEPTH_16U`: enteros de 16 bits sin signo.

  - `IPL_DEPTH_16S`: enteros de 16 bits con signo.
  - `IPL_DEPTH_32S`: enteros de 32 bits con signo.
  - `IPL_DEPTH_32F`: precisión sencilla en flotante.
  - `IPL_DEPTH_64F`: precisión doble en flotante.
- `char colorModel[4]`: ignorado por OpenCV.
- `char channelSeq[4]`: ignorado por OpenCV.
- `int dataOrder`: el orden de los datos. Puede tener los colores almacenados de forma intercalada o separada.
- `int origin`: 0 = origen arriba a la izquierda, 1 = origen abajo a la izquierda.
- `int align`: ignorado por OpenCV.
- `int width`: anchura de la imagen en píxeles.
- `int heigth`: altura de la imagen en píxeles.
- `struct _IplROI *roi`: RegionOfInterest(ROI) región de interés. Si no es `null` solo se trabaja sobre esta región de la imagen.
- `struct _IplImage *maskROI`: debe ser `null`.
- `void *imageId`: debe ser `null`.
- `struct _IplTileInfo *tileInfo`: debe ser `null`.
- `int imageSize`: tamaño de la imagen en bytes.
- `char *imagedata`: puntero a la matriz de datos.
- `int widthStep`: tamaño de una fial en bytes.
- `int BorderMode[4]`: ignorado por OpenCV.
- `int BorderConst[4]`: ignorado por OpenCV.
- `int *imageDataOrigin`: puntero al origen de los datos de la imagen.

`void cvSplit(const CvArr* src, CvArr* dst0, CvArr* dst1, CvArr* dst2, CvArr* dst3)`

Divide una imagen multi-canal en varias imagenes uni-canal.

Parámetros:

- `src`: imagen original.
- `dst0...dst3`: imágenes uni-canal en las que se deposita la información.

`void cvMerge(const CvArr* src0, const CvArr* src1, const CvArr* src2, const CvArr* src3, CvArr* dst)`

Compone una imagen multi-canal con imágenes uni-canal.

Parámetros:

- `src`: imagen original.
- `dst0...dst3`: imágenes uni-canal en las que se deposita la información.

`CvCapture* cvCaptureFromCAM(int index)`

Inicializa la captura de video desde una cámara.

Parámetros:

- `index`: índice de la cámara a utilizar. Si solo hay una cámara o no importa que cámara se desea utilizar se puede pasar como parámetro -1.

`IplImage* cvQueryFrame(CvCapture* capture)`

Adquiere un frame del capturador de una cámara o un fichero y lo devuelve.

Parámetros:

- `capture`: estructura capturadora de video.

`IplImage* cvCreateImage(CvSize size, int depth, int channels)`

Crea una cabecera de imagen y reserva la memoria para los datos.

Parámetros:

- `size`: dimensiones de la imagen.
- `depth`: número de bits con el que se gurada la imagen.
- `channels`: número de canales por píxel.

`void cvReleaseImage(IplImage** image)`

Libera la memoria ocupada por la cabecera de la imagen y los datos.

Parámetros:

- `image`: puntero doble a la cabecera de la imagen.

`void cvReleaseCapture(CvCapture** capture)`

Libera la estructura CvCapture.

Parámetros:

- `capture`: puntero a la estructura capturadora.

`void cvReleaseHaarClassifierCascade(CvHaarClassifierCascade** cascade)`

Libera la estructura CvHaarClassifierCascade.

Parámetros:

- `cascade`: puntero a la estructura CvHaarClassifierCascade.

`CvCapture* cvCaptureFromFile(const char* filename)`

Inicializa la captura de video desde fichero.

Parámetros:

- `filename`: nombre de fichero.

`int cvRound(double value)`

Convierte un flotante en entero redondeando.

Parámetros:

- `value`: valor a redondear.

Bibliografía

[1] http://opencv.willowgarage.com/documentation/index.html.

[2] http://opencv.willowgarage.com/wiki/visualc

[3] http://www.cs.indiana.edu/cgi-pub/oleykin/website/opencvhelp/.

[4] www.wikipedia.org.

[5] Venetsanopoulos. Herodotou, Plataniotis. Automatic location and tracking of the facial region in color video sequences. *Signal Processing: Image Communication*, 1999.

[6] Ivor Horton. *Ivor Horton's Beginning Visual C++ 2005.* Wiley Publishing, 2006.

[7] Gerard Medioni and Sing Bing Kang. *Emerging Topics in Computer Vision.* Prentice Hall Professional Technical Reference, 2004.

[8] Bjarne Stroustrup. *El Lenguaje de Programación C++.* Addison Wesley Longman a Pearson Education Company, 2000.